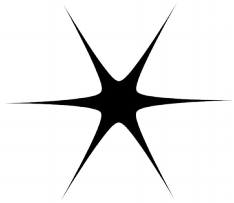


WRAP technology allows you to build POWER-KI language extensions with function libraries that are used by applications in quasi-native mode. That is, it allows you to integrate existing DLLs or your own code in POWER-KI and also push the performance of some tasks to the maximum.

This Guide describes how to use Macros and functions, made available to developers, in the WRAP TEMPLATE VC ++ Project, to build WRAP DLLs.

Once you've mastered the basics, you'll find that with minimal effort you can make your own extensions.

1.0.0

WRAP Developer's Guide

POWER-KI

WRAP

DESCRIPTION			
PROJECT	PROJECT NAME	CODE	XXXXXXXXXX
COVER TITLE	WRAP DEVELOPER'S GUIDE		
FIRST TITLE	POWER-KI	PAGES	24
SECOND TITLE	WRAP	PRINT DATE	
FILE	WRP-DEV-GUIDE-01	SAVED DATE	

MANAGED					
ORGANIZATION	XPLAB	ENTITY	DTC	MANAGE	MXXX.XX

STATUS		
DRAFT	REPLACE	//
	REPLACED	//

DISTRIBUTION	AVAILABILITY	CONFIDENTIAL

IDENTIFICATION								
SUBJECT	OBJ	CAT	T	TYP	ID	SEQ	VER	LANG
PXXX.XX	D	XXX	#	XX	XXXXXXXX	XX	1.0.0	EN

STORAGE						
SUBJECT	OBJ	CAT	T	TYP	ID	SEQ
PXXX.XX	B	XXX	#	XX	XXXXXXXX	XX

REVISION								
MAJOR REVISION HISTORY		CREATED/REVISED			APROVED		
#	NOTE	DATE	BY	NAME	DATE	BY	NAME
0		27/11/20	DTC				

XPLAB s.a.s
viale Sant Eufemia, 39
25135 Brescia – ITALY

Tel. +39 030 2350035

www.xplab.net
www.power-ki.com
www.PowerBerry.tech

Summary

Disclaimer **4**
Document Information **5**
Summary 5
Purpose 5
Validity 5
Relation 5
References 5
Document Change **6**
Terms and Definition **6**
Conventions and Symbol **6**
1 Setting Up the Development Environment **7**
1.1 Get Visual Studio 7
1.2 Get the WRAP template 7
2 Create your Project **8**
2.1 Structure of the Template 9
3 General conventions **10**
3.1 WRAP names 10
3.2 Where to install 10
4 Before You start coding **11**
5 How it Works **12**
6 Understanding PWK and WRAP data **13**
6.1 Data types 13
6.2 PWK_PTR 13
6.2.1 Structure of a PWK_PTR 13
7 Examples **14**
7.1 Simple 14
7.2 Dealing with WRAP PWK_PTR Pointers 15
7.2.1 Creating the pointer 15
7.2.2 Using the pointer 15
7.2.3 Deleting pointers 17
8 Writing Your WRAP **18**
8.1 The Main file 18
8.1.1 PWK-WRP-CORE-01.hpp 18
8.1.2 WRP_DEL 18
8.1.3 WRP_FUNC 18
8.1.4 Functions Init and declaration 18
8.2 Manage PWK_PTR 19
8.2.1 WRP_DCLRES_AS (PWK_PTR as function result) 19
8.2.2 MACRO for Create/Manage PWK_PTR 20
8.3 Trig 21
8.4 WRP_PWK_EXEC 22
8.4.1 WRP_TRACE 22
9 Helpers **23**
9.1 XU_VAL 23
10 Documentation **24**
10.1 Header Table 24
10.2 Use Mode table 24

Disclaimer

While XPLAB sas make every effort to deliver high quality products, we do not guarantee that our products are free from defects.

Our software and documentation are provided “**as is**," and you use the software at your own risk.

We make no warranties as to performance, merchantability, fitness for a particular purpose, or any other warranties whether expressed or implied.

No oral or written communication from or information provided by XPLAB sas shall create a warranty.

Under no circumstances shall XPLAB sas be liable for direct, indirect, special, incidental, or consequential damages resulting from the use, misuse, or inability to use this software, even if XPLAB sas has been advised of the possibility of such damages.

Document Information

Summary

Describes the WRAP VC++ Template, its installation and use for the creation of WRAP DLL.

Purpose

Providing the basic information about POWER-KI WRAP technology.

Validity

Applicable to PWK ver. 10 Codename GLUE.

Relation

POWER-KI programming manuals.

References

[1] POWER-KI A PROGRAMMING LANGUIAGE
Preludio
Cesare A. Perani
2012 - XPLAB

Document Change

Terms and Definition

Glossary entry	Entry definition
PWK	POWER-KI
1bsd	ONE based (in opposition to 0bsd) as start index
PWK_PTR	Pointer to a PWK envelope of application pointer

Conventions and Symbol

Text	Description	Example
`Courier new`	Code or code symbol	`U8 s=10;`

1 Setting Up the Development Environment

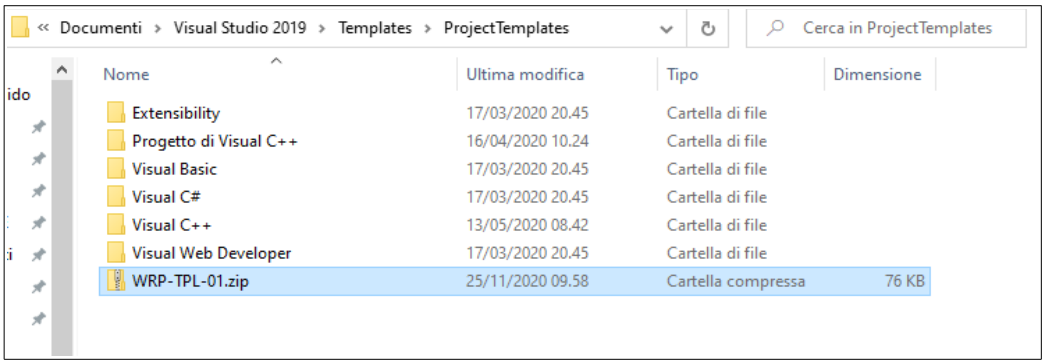
The steps are:

1.1 Get Visual Studio

From Microsoft site download and install Visual Studio 2019, from VS installer make sure to have selected "Desktop Application Development with C ++".

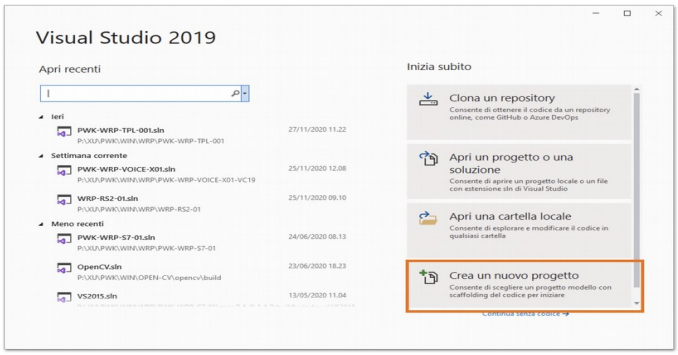
1.2 Get the WRAP template

From "https://github.com/POWER-KI/POWER-KI/tree/master/WRAP/TEMPLATE-DLL"
download "WRP-TPL-01.zip".
Copy the Template in your folder: "Documents\Visual Studio 2019\Templates\ProjectTemplates".

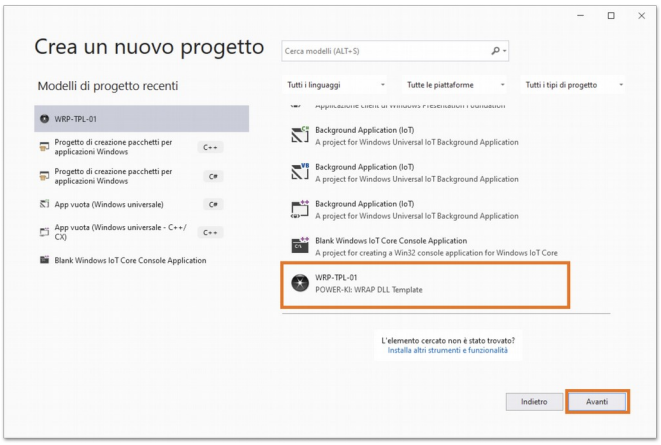


2 Create your Project

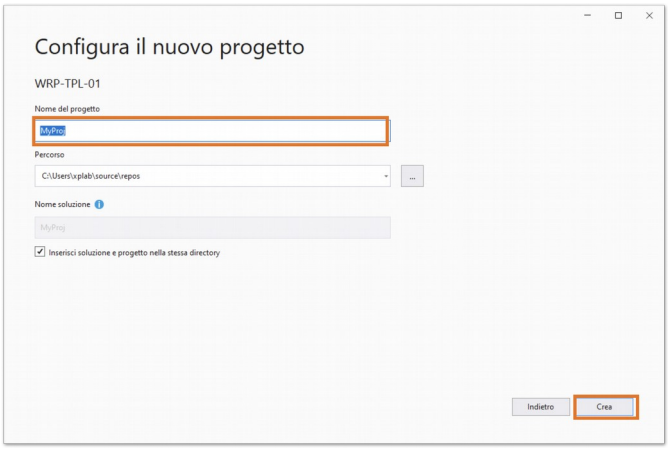
Start VS and choose "create a new project":



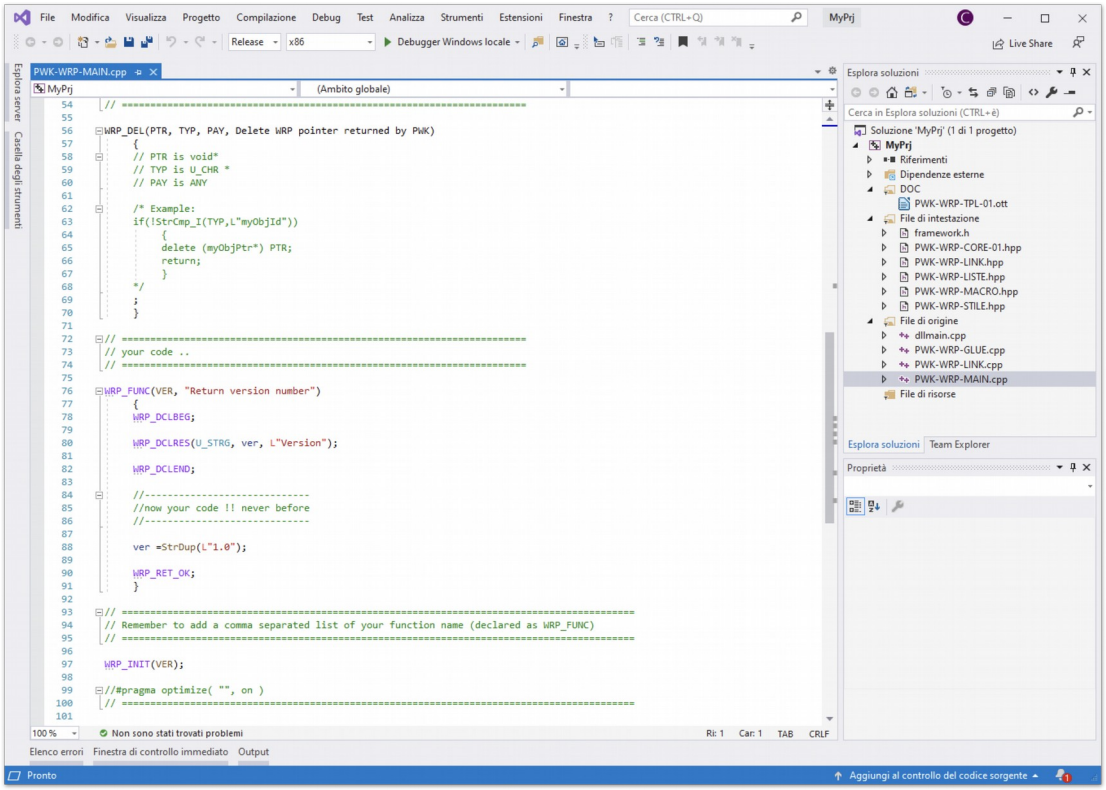
Scroll down and search and select the Template:



Choose the Project name and the destination folder then press [CREATE]



2.1 Structure of the Template



The new clean project derived from the Template contains a series of header and code files.

Among the latter, the one you will need to use to define ALL the functions of your library that must be visible by POWER-KI (PWK) is "PWK-WRP-MAIN.cpp" (to which you can change the name if you want).

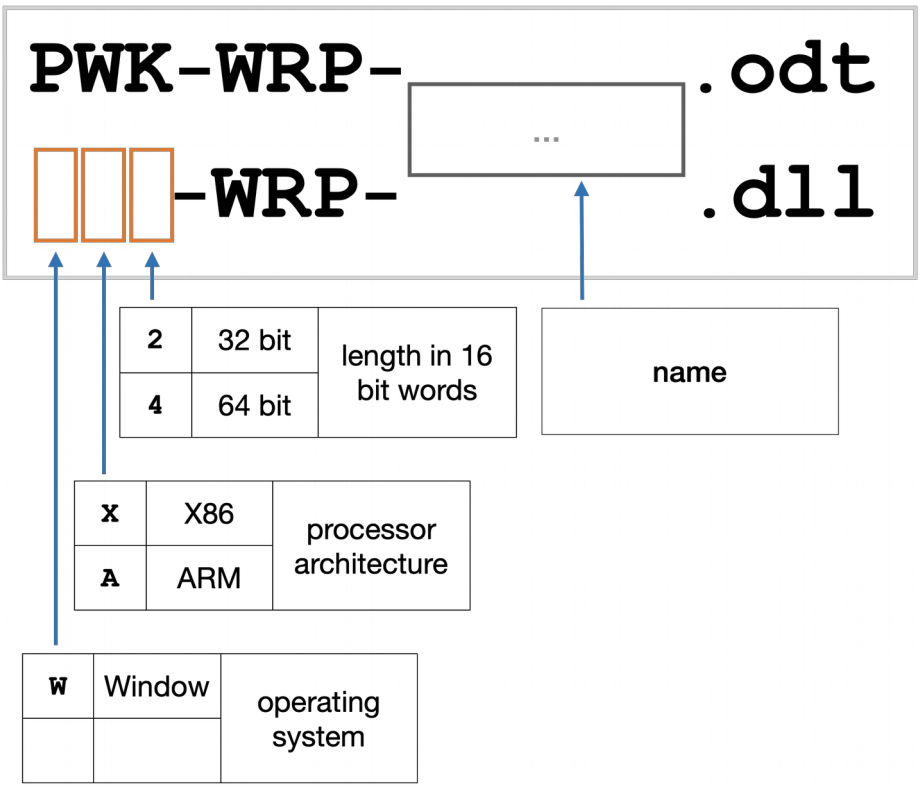
The other files contain both the functions for connecting the DLL to PWK and those that facilitate the exchange and conversion of data. The "DOC" section contains the ".odt" template for creating the function documentation file, which will be used by PWK in the EditorAssistant and by the FunctionComposer.

3 General conventions

Your WRAP DLL to be loaded by PWK must respect some conventions.

3.1 WRAP names

The name of the DLL and the associated documentation file (.odt) must comply with the following scheme:



.dll and .odt files must have the same name. Template is set to generate the .dll name starting from the project name: WX2-WRP-*ProjectName*.dll .

3.2 Where to install

Install procedure means simply put the .dll and .odt in the same place that could be:

WRAP folder in PWK install directory	used for WRAPs included in PWK distributions
WRAP folder in PWK-PRG directory	for user general visible WRAPs
In the application Package	for only application use (best if you plan to distribute your app).
in the application directory	deprecated

If you have added the WRAP to the package: after the inclusion you should close and reopen it to get the WRAP working, if instead you want to change an existing WRAP you must first remove it from the package, close and reopen the package then add the new one and again close and reopen the package.

4 Before You start coding

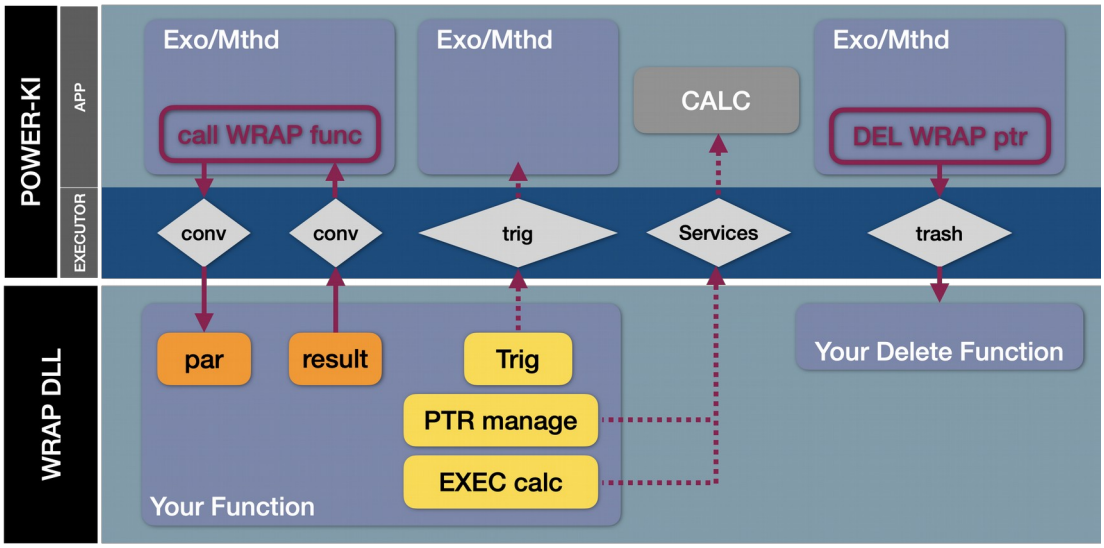
Before you start writing your code, we believe it is important to clarify some concepts in order to respect the spirit of POWER-KI.

You are what in XPLAB we call the "Core Programmer" (CP) and your library will be used by what we call "Application Programmers" (AP).

First of all, when designing your functions you will have to imagine to be the AP that has to use them, so in principle:

- their purpose must be clear;
- they must not be too numerous, possibly grouping those with similar purposes in a single function and using the parameters to discriminate the context of use;
- include in functions, calls to those functions that should or may need to be performed previously (eg if a function on images requires a gray scale image to be passed, but the application may originally have a colored image, the function must be able to accept the latter and in this case carry out the conversion automatically) ;
- consider that if, due to an incorrect parameter given to your function by the AP. the program crashes, in the spirit of PWK the error is of the CP and must be corrected, that is: the program must be resilient;
- the tools that PWK provides to document functions are very powerful .. use them!

5 How it Works



From your PWK App you can call your DLL function, the call parameters are converted according to the types of your wrap declarations, the WRAP function is executed and the control revert to the PWK caller with the result parameter (if declared) even it converted according to its declaration.

To get intermediate results or status or operation to perform during WRAP function execution, the PWK Caller can provides (as first parameter) a PWK_PTRto a TRIG that the function can call when needed.

PWK makes available to the WRAP services in form of function for managing PWKPTR, and for the execution of PWK Calc.

Inside the APP, the deletion of WRAP PTR, that is of PTR that encapsulate WRAP object, is managed by PWK invoking the delete function of your WRAP.

6 Understanding PWK and WRAP data

In PWK the attributes (data) do not have a defined type, but C ++ does. The functions contained in the Template allow you to easily manage these exchanges.

6.1 Data types

In the following table the definition of the data types that you can use to define the parameters and return types (PAR) or inside helper functions.

WRAP	As Par	C++ (32 bit)	Note
I8	YES	char	
U8	YES	unsigned char	
I16	YES	short	
U16	YES	unsigned short	
I32	YES	int	
U32	YES	unsigned	
I64	YES	long long	
U64	YES	unsigned long long	
F32	YES	float	
F64	YES	double	
U_STRG	YES	(U_CHR*)	
ANY	YES	(void*)	
U_CHR	**NO**	wchar_t	
A_CHR	**NO**	char	
XU_VAL	**NO**	(U_CHR*)	Is a C++ Object

6.2 PWK_PTR

In PWK there are functions to create and manage PWK_PTR to encapsulate object in pointers (e.g. BUF, LIS), even your WRAP may have this need so the Template provides functions for their generation and management.

6.2.1 Structure of a PWK_PTR

PWK_PTRhas five elements:

PTR	LIB	TAG	TYP	PAY
ANY	U_STRG	U_STRG	U_STRG	ANY

The PTR is the pointer to the encapsulated object;

the LIB is the library under which the PTR belong (it creates and delete it);

the TAG is a text defined by the lib (mandatory);

the TYP is a text defined by the lib (optional);

the PAY is defined by the lib (optional).

Pointer that encapsulate WRAP object that must be deleted by the WRAP should be declared with LIB=WRP and TAG=EXT, PWK will then substitute the LIB with the lib name.

7 Examples

7.1 Simple

```
WRP_FUNC(VER, "Return version number")
        {
        WRP_DCLBEG;

        WRP_DCLRES(U_STRG, ver, L"Version");

        WRP_DCLEND;

        //-----------------------------
        //now your code !! never before
        //-----------------------------

        ver =StrDup(L"1.0");

        WRP_RET_OK;
        }

WRP_FUNC(SUM, "Return the sum of two I32")
        {
        WRP_DCLBEG;

        WRP_DCLPAR(I32, a1, L"First addendum");
        WRP_DCLPAR(I32, a2, L"Second addendum");

        WRP_DCLRES(I32, res, L"result");

        WRP_DCLEND;

        //-----------------------------
        //now your code !! never before
        //-----------------------------

        res =a1+a2;

        WRP_RET_OK;
        }

// ================================================================================
// Remember to add a comma separated list of your function name (declared as WRP_FUNC)
// ================================================================================

WRP_INIT(VER,SUM);
```

In the example we see the definition of two functions `WRP_FUNC(VER)`, `WRP_FUNC(SUM)` and their declaration `WRP_INIT(VER,SUM)`.

`WRP_FUNC(VER)` declares an `U_STRG` as return result, in this way the returned version can contains not only number, the helper `StrDup` function is used, that return a new allocated pointer to a UNICODE string that contains a copy to the `L"1.0"` parameter.

`WRP_FUNC(SUM)` declares two `I32` input parameter `a1,a1` and an `I32` as result that will return the sum of `a1+a2`.

Both the functions are then declared in **`WRP_INIT`** comma separated list, to make them visible to PWK,

Assuming MyPrj as lib name, here an example of how you can use them in PWK CODE:

```
chatput(MyPrj_VER);      !!  1.0 will be printed on the chat;
chatput(MyPrj_SUM(1,3));!!  4 will be printed on the chat;
```

7.2 Dealing with WRAP PWK_PTR Pointers

This is an excerpt from OpenCv WRAP DLL.

7.2.1 Creating the pointer

With the function `VCP` we create WARP PWK_PTRto a video capture:

```
WRP_FUNC(VCP, "Create Video Capture ")
        {
        WRP_DCLBEG;

        WRP_DCLPAR(U_STRG, wht, Image source name);
        WRP_DCLPAR_DEF(int, api,0);

        WRP_DCLRES_AS(RES, L"VCP");

        WRP_DCLEND;

        //----------------------------
        //now your code !! never before
        //----------------------------

        RES = CreateVcp(wht, int(api));

        if (RES)
                {
                WRP_RET_OK;
                }

        WRP_RET_ERR;
        }
```

As input parameter we declare the `U_STRG wht` that contains the source of the image (can be the cam number of the pc or the internet address of a stream), the second parameter is optional and if not defined by the caller is set to 0. As result with **`WRP_DCLRES_AS`** we declare a WRAP pointer with TYP `L"VCP"` (as we declare only one element. the type, PWK set by default the LIB to `WRP` and the TAG to `EXT`).

If `RES` as result of the internal defined WRAP function `CreateVcp` is valid, the function will end with **`WRP_RET_OK`** otherwise with **`WRP_RET_ERR.`**

7.2.2 Using the pointer

From PWK we can now invoke the WRAP function to get the PWK_PTR to the VideoCapture Object:

```
Code  84  insert area
vcp=ocv_vcp(0); !!open the vcp on the pc cam;
```

and use it to get the image :

```
Code  88  insert area
mat=ocv_vcp_rd(vcp); !!Get the image as ocv mat;

!!and send it to the stream;
gui_STREAM(_PTR_@\MainGui, $\MainGui\pag\PANEL_name\STREAM,mat);
```

The WRAP function `ocv_vcp_rd` is defined in this way:

```
WRP_FUNC(VCP_RD, "Read from Video Capture ")
        {
        WRP_DCLBEG;

        WRP_DCLPAR(ANY, PTR, Video capture PTR);

        WRP_DCLRES_AS(RES, L"MAT");

        WRP_DCLEND;

        //-----------------------------
        //now your code !! never before
        //-----------------------------
        if(_parNum_<1)
                {
                WRP_RET_ERR;
                }

        PTR_TO(OCV_VCP)ocv= (OCV_VCP*) GET_OCV_PTR(PTR, L"VCP");

        if (!ocv)
                {
                WRP_RET_ERR;
                }

        cv::Mat m;
        if(ocv->vcp.read(m)==0)
                {
                WRP_RET_ERR;
                }

        if(m.empty())
                {
                WRP_RET_ERR;
                }

        RES = CreateMat(m);

        WRP_RET_OK;
        }
```

A PWK_PTR of typ VCP is required to read from OpenCv with `ocv->vcp.read` if the imagine is valid, an object is created with the user function CreateMat and then returned a PWK_PTR of type MAT to PWK.

With the user function `GET_OCV_PTR` the PTR parameter is checked for existence and equality to VCP and if the condition is satisfied the embedded ptr is returned:

```
ANY GET_OCV_PTR( ANY p, PTR_TO(U_CHR)ty)
        {
        XU_VAL LIB,  TYP;

        if (p)  {
                LIB = (U_CHR*)WRP_PTR(p, LIB);
                TYP = (U_CHR*)WRP_PTR(p, TYP);

                if (LIB == L"OCV" && TYP == ty)
                        {
                        return WRP_PTR(p, PTR);
                        }
                }
        return NULL;
        }
```

In this user defined function, you can see the use of the provided helpers `XU_VAL` objects for UNICODE string manipulation, and `WRP_PTR` for PWK_PTR manage.

7.2.3 Deleting pointers

From PWK CODE WRAP PWK_PTR can be deleted as any other pointers:

What happen under the hood is that the `WRP_DEL` function of the LIB is invoked.

Using the TYP as selector you can delete the right object:

```
WRP_DEL(PTR, TYP, PAY, Delete WRP pointer returned by PWK)
        {
        // PTR is void*
        // TYP is U_CHR *
        // PAY is ANY

        if(!PTR)return;
        XU_VAL msg;

        switch(StrSelect(TYP, L"MAT,VCP,VWR,BKS,CCL"))
                {
                default:        msg= L"OCV DEL - Unknown Typ: ";
                                msg <<= TYP;
                                WRP_TRACE(msg);break;

                case 1: delete (OCV_MAT*)PTR; break;
                case 2: delete (OCV_VCP*)PTR; break;
                case 3: delete (OCV_VWR*)PTR; break;
                case 4: delete (OCV_BKS*)PTR; break;
                case 5: delete (OCV_CCL*)PTR; break;
                }
        }
```

To be noted in this code the use of the helpers: `StrSelect` that search a symbol inside a string of comma separated value and if found returns its position (1bsd), and `WRAP_TRACE` that print a text in PWK chat (using `chatput`, useful for debug purpose).

8 Writing Your WRAP

PWK-WRP-MAIN.cpp	
`#include <PWK-WRP-CORE-01.hpp>`	Mandatory
`WRP_DEL``(PTR, TYP, PAY, Delete WRP pointer returned by PWK)`	
`WRP_FUNC``(xxx, "...")`	**`WRP_DCLBEG;`** Mandatory
	`WRP_DCLPAR, WRP_DCLPAR_DEF` input par
	`WRP_DCLRES, WRP_DCLRES_AS` result
	`WRP_DCLEND;` Mandatory
	`your code`
	`WRP_RET_OK, WRP_RET_ERR` Mandatory (one)
`WRP_INIT``(xxx,...)`	Mandatory

8.1 The Main file

The default name is `PWK-WRP-MAIN.CPP` but you can change it.

8.1.1 PWK-WRP-CORE-01.hpp

Contains the macro and the reference to the helper functions necessary or useful to connect the WRAP to PWK . Should be included directly or indirectly.

8.1.2 WRP_DEL

If the WRAP create and pass to PWK with **WRP_DCLRES_AS**, its own type of PWK_PTR, when thy are trashed this function is invoked by PWK and the three parameters are:

`PTR`	ANY	Pointer of the embedded WRAP object declared by `WRP_DCLRES_AS`
`TYP`	U_STRG	the Declared TYP in `WRP_DCLRES_AS`
`PAY`	ANY	the optional payload

8.1.3 WRP_FUNC

All the function that are called by PWK must be defined in this way.

The very first thing that must be defined are the input parameters (if any). and the result to be returned (if any). These must be included between the **`WRP_DCLBEG`** and **`WRP_DCLEND`** keywords which must be there in any case. With `WRP_DCLRES` you can declare any number of parameters but only the first one is returned as result to PWK. The reason you might declare multiple `DCLRES` is that they can be used as trigger parameters. Every return path of the function should be closed by one of:

`WRP_RET_OK`	Success	if declared the WRP_DCLRES is returned, else 1 is returned
`WRP_RET_ERR`	Fail	NULL is returned

8.1.4 Functions Init and declaration

In **WRP_INIT(…)** as parameters put the comma separated list of all the `WRP_FUNC` names that you want to use in PWK CODE (in PWK they will be in the form: *`LibName_FuncName)`*.

8.2 Manage PWK_PTR

As seen in the examples, PWK_PTRs are essential elements for the exchange of complex information, as they allow to encapsulate pointers to objects or structures in a tagged envelope. PWK_PTRs used by PWK such as `BUF` and `MTX` are definable and manageable within WRAP, but WRAPs can also define their own PWK_PTRs, which can then be passed by PWK as parameters between WARP functions and eventually destroyed consistently.

8.2.1 WRP_DCLRES_AS (PWK_PTR as function result)

Using WRP_DCLRES_AS you can tell to PWK to create a PWK_PTR as return result of your function. The parameter of the macro are the name of the variable that will contains the pointer to the object to embed and the string `AS`.

Inside the WRAP you can create these PWK_PTR of PWK object:

PTR	AS	LIB	TAG	TYP	PAY	note
ptr to `U8`	`L"BUF:BUF::x"`	BUF	BUF	//	x=size in byte	when deleted also PTR is deleted
ptr to `U8`	`L"BUF:CNK::x"`	BUF	CNK	//	x=size in byte	when deleted ptr is NOT deleted
ptr to `BUF_MTX_DATA`	`L"BUF:MTX"`	BUF	MTX	//	//	`BUF_MTX_DATA` is defined in PWK-WRP-LINK.hpp

To create your WRAP PWK_PTR you can use the IMPLICIT form that is putting in `AS` only the TYP.

With the EXPLICIT form you should set LIB=`WRP` and TAG=`EXT`, on return from the WRAP function, PWK will change LIB to your lib name.

If `AS` is void no PWK_PTR is created and the values is managed as `ANY`.

`WRP_DCLRES_AS` creates a new PWK_PTR so after the creation, passing the PWK_PTR to and back from functions should be made with `ANY` parameter/result type.

As `WRP_DCLRES_AS` can only be used at the begin of the function (between `WRP_DCLBEG` - `WRP_DCLEND`) there are situations in which either not all the data is known or it is necessary to change some element, to do that you can use the provided macro:

`WRP_SETRES_AS`	Replace the entire AS string
`WRP_SETRES_AS_PAY`	Replace AS with the new AS:PAY (`unsigned`)
`WRP_SETRES_AS_TYP`	Replace AS with the new AS:TYP (`U_STRG`)

`WRP_GETAS`	Return the AS (`U_STRG`) associated to the parameter

8.2.2 MACRO for Create/Manage PWK_PTR

Inside your WRAP you may have the need to mange PWK_PTR that are non `DCLRES`, the Template provides helper MACRO to to so:

`WRP_PTR_NEW(PTR,AS)`

Create a new PWK_PTR(do not use for your WRAP PWK_PTR) for the PTR object.

Return `ANY` or `NULL` if error.

`WRP_PTR_DEL(PWK_PTR)`

Delete a PWK_PTR,

`WRP_PTR(PWK_PTR,WHT)`

Return according with WHT (use WHT directly e.g. `WRP_PTR(myPtr,LIB)`) :

PTR → (`ANY`) PTR;

LIB → (`U_STRG`) the LIB;

TAG → (`U_STRG`) the TAG;

TYP → (`U_STRG`) the TYP;

PAY → (`ANY`) PAY;

`WRP_PTR_SET(PWK_PTR,WHT,p1)`

Set the element of AS indicated by WHT to p1.

`WRP_PTR_LCKS(PWK_PTR)`

Lock by one the SEMAPHORE of PWK_PTR, return the counter of the locks.

`WRP_PTR_LCKR(PWK_PTR)`

UnLock by one the SEMAPHORE of PWK_PTR, return the counter of the locks.

`WRP_PTR_TRY(PWK_PTR)`

Try to lock the SEMAPHORE of PWK_PTR, return 1 if success.

8.3 Trig

During its execution a function can start the execution of a TRIG PWK, this must be created in PWK and passed as the first parameter in the function call. The TRIG must not be included among the parameters declared by the function and will be called automatically by the appropriate functions. The TRIG can be executed in SYNCHRONOUS or ASYNCHRONOUS mode, with or without parameters according to the MACRO with which it is invoked. The call parameters that can be used are those declared with `WRP_DCLPAR` or `WRP_DCLRES` (the fact that the WRAP function returns only the first first declared result, but allows you to declare others is functional for this purpose).

`WRP_TRIGSYNC`

The TRIG is executed without parameters synchronously.

`WRP_TRIGPARL`

The TRIG is executed without parameters in parallel.

If with parameters, their names must match the names declared in the TRIG.

`WRP_TRIGSYNC_With(...)`

The TRIG is executed with parameters synchronously.

`WRP_TRIGPARL_With(...)`

The TRIG is executed with parameters in parallel.

To synchronize the access to the trig (if needed) use:

`WRP_TRIG_LCKS` to Lock;

`WRP_TRIG_LCKR` to UnLock.

8.4 WRP_PWK_EXEC

With `WRP_PWK_EXEC` the WRAP has a way to execute a POWER-KI code inside the PWK App and get back a result.

In the following example, an excerpt from RS2 WRAP, the function call PWK for the execution of the conversion of an OpenCv MAT to a different type (`CV_64F`) and the execution of `OCV_OP`.

```
// Converts depth frame to a matrix of doubles with distances in meters
ANY depth_frame_to_meters(const rs2::pipeline& pipe, const rs2::depth_frame& f)
        {
        using namespace rs2;
        U_CHR buf[BFsiz];

        ANY dm = PWK_frame_to_mat(f);
        ANY RES=NULL;

        if(!dm) return NULL;

        FLT depth_scale = pipe.get_active_profile()
                .get_device()
                .first<depth_sensor>()
                .get_depth_scale();

        StrFormat(buf, BFsiz,
        (U_CHR*)L"°t=%s;°t1=OCV_MAT(°t,£CNV,£CV_64F);°t2=OCV_OP(°t1, £MUL,%f);TRASH(°t,°t1);°t2",
         dm, depth_scale);

        delete dm;
        RES = WRP_PWK_EXEC(buf);
        return RES;
        }
```

8.4.1 WRP_TRACE

The Macro `WRP_TRACE(L"some text")` print the parameter (`U_STRG`) on the chat of the PWK app. It is useful for WRAP debug.

9 Helpers

In POWER-KI attributes (VARIABLES data) are in form of UNICODE strings, so to simplify the exchange to and from, the template provides a series of functions declared in the PWK-WRP-LINK.hpp, in addition to this there is the XU_VAL object.

9.1 XU_VAL

XU_VAL are objects for the management of U_STRG whose content can derive from the assignment or copy of U_STRG, from the content or from the copy of the content of another XU_VAL, or from numerical values. If their content is a numeric value (expressed as a string) is possible to extract it with simple cast operations.

```
XU_VAL v1;
XU_VAL v2;
XU_VAL v3;

v1=L"Test";              // Copy of the U_STRG constant
v2.SetS(v1.Reset());     // The content of v1 is transferred to v2 and v1 is cleared
v3.SetS(StrDup(L"Alpha"));

v1=L"A+100";
v2.SetS(v1.GetUpToChar(L'+');    // v2=L"A"
v3.SetS(v1.GetFromChar(L'+');    // v3=L"100"

int n=(int)v3;  //n=100

v1=v3;           //v1=L"100"
v1 <<= L"-";     //v1=L"100-"
v1 <<= v2;       //v1=L"100-A"

v1=100;
v2=2;
v3=100*2;        //v3=L"200";

v3 <<= L"kg";    //v3=L"200kg";

//with XU_VAL_TMP an U_STRG, temporary behaves as a regular XU_VAL

U_STR ps=L"TEST";
int tr= XU_VAL_TMP(ps)==L"TEST";// tr now is TRUE
```

10 Documentation

The Template includes the document template to be used to generate the function documentation file, which can be used by the EditorAssistant and the POWER-KI FunctionComposer.

A level 2 chapter must be created for each function, named according to the following scheme:

MyLib_FuncName

where the *MyLib* is the name of your lib and *FuncName* is the name that you have declared `WRP_INIT`.

The chapter must contains at least two tables.

10.1 Header Table

xxx_VER	fnc
Description	Return the dotted version number
Related	
Remark	

Do Not change it schema or row header text.

10.2 Use Mode table

For each variant of use of the function, with progressive numbering (starting from 1), there must be a usage mode table like:

1.xxx_VER					
Parameter	**Type**	**Values**	**Comment**	**Default**	**Opt**
Return					
VER	symb		Version Number		
OnError					
Example					

In this table you can add rows for parameters, Return and OnError but you can't change column names or schema.

In right top field you can put a description of the variant.

As PARAMETER you can put an identifier (a free name).

TYPE is free but as practise we use one of; symb, NV, enum, slis.

If the user has to choose from a list of values, put them in VALUES followed by an optional explanation contained in round bracket, separating each element by a semi colon:

£TRUE(if true);£False(if false)

COMMENT and DEFAULT are freeText, in OPT put YES if the field is optional.

For examples you can take a look to POWER-KI manuals.